A Simple Refinement of Slow-start of TCP Congestion Control

Haining Wang† Hongjie Xin‡ Douglas S. Reeves‡ Kang G. Shin†

†RTCL, EECS Department
The University of Michigan
Ann Arbor, MI 48109
{hxw,kgshin} @eecs.umich.edu

‡Department of Computer Science
North Carolina State University
Raleigh, NC 27695
{hxin,reeves} @eos.ncsu.edu

Abstract

This paper presents a new variant of Slow-start, called Smooth-start, which provides a smooth transition between the exponential and linear growth phases of TCP congestion window. Slow-start is known to make an abrupt transition between the Slow-start and Congestion-Avoidance phases, and hence, often causes multiple packet losses from a window of data and retransmission timeouts, which, in turn, reduce effective throughput and result in global synchronization. Smooth-start solves this problem by approaching the Slow-start threshold more gradually. Our extensive simulation results show that Smooth-start can significantly reduce both packet losses and traffic burstiness, thus improving the performance of TCP congestion control at the start of a TCP connection or after a retransmission timeout. Furthermore, Smooth-start is very simple to implement and requires TCP modifications at the sender side only.

1 Introduction

The wide use of the TCP/IP protocol suite and the explosive growth of the Internet have made TCP congestion control crucial to the performance of the Internet. The current implementation of TCP congestion control includes the classic algorithms of Slow-start and Congestion Avoidance [8], and the augmentation of Fast Retransmit and Fast Recovery algorithms [16]. Many popular Internet application protocols, such as HTTP, ftp, and telnet, are implemented with TCP. Since these protocols generate a large percentage of traffic on the Internet, the TCP congestion control should be optimized by adapting itself to the common behavior of these protocols.

The recent Internet traffic measurements [17] indicate that many TCP flows are short-lived but most of TCP packets in flight still belong to long-lived TCP flows. In contrast with the long-lived transfers, the characteristics of the short transfers are that a relatively small number of data packets are delivered and a TCP connection is usually terminated before it reaches steady state. The popularity of the Internet has increased the complexity and size of networks, which makes it take longer to probe appropriate control parameters of a TCP connection. Thus, the start-up period of a TCP connection can greatly influence the performance of short-lived TCP connections. Also, the increasing demand for network resources has made bursty packet losses common, which often leads to a retransmission timeout. So, the re-start-up period of a TCP connection, plus the start-up period, can significantly affect the performance of long-lived TCP connections.

TCP Slow-start is initiated both at the start of a TCP connection or after a retransmission timeout, and hence greatly influences the performance of short-lived and long-lived TCP connections. The objective of Slow-start is to enable a TCP sender to discover the available network bandwidth by gradually increasing the amount of data injected into the network from an initial window size of one segment, which prevents the TCP sender from congesting the network with a large burst of data. Unfortunately, the approach that Slow-start uses to probe the network bandwidth is counter-intuitive. It drizzles data out at the beginning, and ends with a drastic leap to reach the Slow-start threshold. If we fill a pipe or a container with water, the approach we take is opposite to the way Slow-start works.

Due to the way Slow-start probes the network bandwidth, it suffers from the following two problems. First, since the Slow-start algorithm begins with sending one segment, it takes many round-trip times to reach the optimal operating point, thus resulting in poor utilization of the available bandwidth for short transfers which are small compared to the bandwidth-delay product of the path. Second, since a TCP sender has no knowledge about the capacity of the available resources on the networks and uses default parameters at the beginning of transmission, the exponential growth of the congestion window often misleads the sender to send too many packets too quickly, thus causing a severe buffer overflow at the bottleneck link. This buffer overflow results in multiple packet losses from a window of data, thereby making the TCP senders lose their self-clocking. The subsequent retransmission timeouts cause a global synchronization [15], [20]. The global synchronization lowers the aggregate throughput and makes the network traffic load and the queueing delay oscil-

0-7695-0722-0/00 $10.00

late, and hence, the TCP performance degrades substantially.

To resolve the first problem, a larger initial window and TCP Fast-start have been proposed [1], [3], [12] to speed up the transfer rate at the very beginning of Slow-start, which greatly benefits short TCP transfers. To efficiently recover multiple packet losses within a window of data, New-Reno and SACK [7], [10] have been proposed to recover from bursty packet losses without losing self-clocking. Furthermore, to remove the second problem of Slow-start, estimated initial *ssthresh* and safe *ssthresh* [7], [19] have been proposed to replace the default setting of *ssthresh*. However, the abrupt transition of congestion window between exponential growth and linear growth phases that causes highly bursty traffic and frequent buffer overflows, remains unaddressed.

Although the estimated initial *ssthresh* and safe *ssthresh* can make significant improvements over an arbitrary default *ssthresh*, there are still several obstacles that make it very difficult for the sender to estimate the congestion control parameters in an accurate and timely manner: (1) ACK compression [20], which makes the pacing between ACKs not reflect the bottleneck delay experienced by the data packets; (2) the delay caused by the receiver to generate an ACK for each newly-arrived data packet; (3) routes are often asymmetric, which can lead to bandwidth and latency asymmetries; (4) with a varying number of connections, the remaining network capacity along the path also varies over time; (5) out-of-order packet delivery, which occurs frequently; (6) multi-channel bottleneck links, which violate the assumption that there is a single end-to-end forwarding path. Therefore, it is very difficult to set up and maintain an appropriate *ssthresh*. The improvement of sender-side estimated *ssthresh* is thus limited. A receiver-side estimation algorithm has been proposed [2], and shown to yield better bandwidth estimation than the sender-side estimation. However, it requires to modify the receiver side, which results in a scalability problem in the context of millions of clients in the Internet.

In this paper, we propose a modification of Slow-start, called *Smooth-start*, which is orthogonal to the proposals for selecting an appropriate initial *ssthresh*. The objective of Smooth-start is to make a smooth transition between the exponential and linear growth phases of the congestion window by changing the way the TCP sender uses to reach the Slow-start threshold. A separator *smsthresh* is introduced, which is set to $ssthresh/2^d$, where d is a non-negative integer. As the window size becomes larger than *smsthresh*, we slightly reduce the acceleration rate of the congestion window. The interval [*smsthresh*, *ssthresh*] is called the *Smooth-start period*. During this period, the window size still increases exponentially, but at a reduced rate. Therefore, the aggressiveness of the congestion window's growth during the Smooth-start period is reduced, thus reducing packet losses and dampening traffic burstiness. Smooth-start brings several benefits, even without the *ssthresh* estimator or when the estimated *ssthresh* is inaccurate or out of date.

- Smooth-start produces less bursty traffic than Slow-start, which reduces the fluctuation of the offered load on the networks.
- With the help of early congestion signals, the Smooth-start reduces the chance of buffer overflow at intermediate routers.
- The reduced chance for buffer overflow also reduces the chance for multiple packet losses within the same window.
- Without the help of early congestion signals, the Smooth-start delays congestion. This delayed congestion benefits short-lived connections because they often finish transmission even before the congestion occurs.

Moreover, the implementation of Smooth-start is very simple and its overhead is very small. Only the sender side requires modifications, thus facilitating incremental deployment in today's Internet. Also, due to the inherent conservativeness of Smooth-start, it doesn't hurt fairness. Since Smooth-start is triggered at the beginning of a connection or after a retransmission timeout, it benefits not only short-lived TCP transfers but also bulk TCP transfers if a congestion occurs.

The drawback of Smooth-start is that it takes longer for the transmission rate to reach the optimal equilibrium operating point. However, the number of extra round-trip times introduced by Smooth-start is small and deployment of a larger initial window or TCP Fast-start, which increases the growth rate of the congestion window at the first round-trip time of Slow-start, compensates the slower growth rate of the congestion window in the Smooth-start period.

The remainder of this paper is organized as follows. Related work is described in Section 2. Section 3 briefly introduces Slow-start while Section 4 presents its details as well as the possible probe strategies. Section 5 presents the simulation results for constant- and changing- load experiments. Finally, Section 6 concludes the paper.

2 Related Work

A larger initial window [1] has been proposed to enhance the TCP performance, which raises the initial window size from 1 MSS to 4 Kbytes. Evaluation of a larger initial window [3] shows that it decreases the transfer time of short-lived TCP over dialup channels and the Internet, and also the larger initial window does not significantly increase the number of retransmitted packets. To speed up Web transfers, TCP Fast-start [12] reuses cached network parameters in the recent past to shorten the startup of Slow-start, which greatly reduces transfer latency for short bursts. Also, by assigning a higher drop priority to Fast-start packets and augmenting the TCP loss recovery, TCP Fast-start tries to avoid performance degradation when the cached information is stale.

An appropriate initial value of *ssthresh*, which dictates when to switch from the Slow-start to Congestion-Avoidance phase, is important to the performance of a TCP connection. Recognizing this importance, new approaches [7], [19] have been proposed to replace the default setting of *ssthresh* with an estimated or safe value of *ssthresh*. The estimation method in [7] is similar to the *packet-pair* technique proposed in [9]. In [19], a new variant of SPAND (Shared PAssive Network Discovery [14]) has been proposed to extract the current network condition, and based on the extraction, to derive optimal initial TCP parameters. However, these sender-side bandwidth estimation schemes are found to be problematic in practice. A receiver-side algorithm has been developed to estimate the available bandwidth more accurately [2].

A modified Slow-start algorithm is introduced in TCP Vegas [5], which limits the exponential growth of Slow-start to every other round-trip time, instead of every round-trip time. However, the problem is that multiple packet losses from the same window may happen during the exponential growth phase in the modified Slow-start. The key part of TCP Vegas [5] is the modified congestion avoidance algorithm, which allows more efficient network bandwidth utilization and higher network throughput.

3 Slow-Start

In the original TCP specification [13], the window used by the sender, denoted as *wnd*, is equal to the receiver advertised window *rwnd* regardless of the load in the network. However, in the TCP congestion control schemes initiated by Van Jacobson [8], the TCP window size *wnd* is set to the minimum of the congestion window and *rwnd*. The congestion window *cwnd* is adjusted dynamically in response to network congestion.

$$wnd = min(cwnd, rwnd)$$

The TCP congestion control algorithm runs in two phases: Slow-start and Congestion-Avoidance. When $cwnd < ssthresh$, the algorithm is in the Slow-start phase. Every received acknowledgment increases *cwnd* by 1. During this phase, *cwnd* exponentially increases at every round-trip time.

After *cwnd* reaches *ssthresh*, the congestion control algorithm is in the Congestion-Avoidance phase. Every received acknowledgment increases *cwnd* by $1/cwnd$. During this phase, *cwnd* linearly increases at every round-trip time.

If the TCP receiver acknowledges each received data packet and no congestion occurs during the Slow-start phase, the amount of time required for *cwnd* to reach *ssthresh* is as follows:

$$Slow-start\ Time = RTT \times \log_2 ssthresh$$

The fundamental problem in the Slow-start algorithm is that when the congestion window size approaches close to the equilibrium of the connection, the increment of the congestion window size is too large, causing packet loss. At the beginning of the transmission, since the network is empty, the exponential increase of congestion window size is reasonable and also necessary to fill the empty pipeline quickly. However, as *cwnd* gets close to the equilibrium of the connection (unfortunately the sender does not know this until packet loss occurs), the scheme of *one ACK causing one increment of congestion window size* becomes too aggressive.

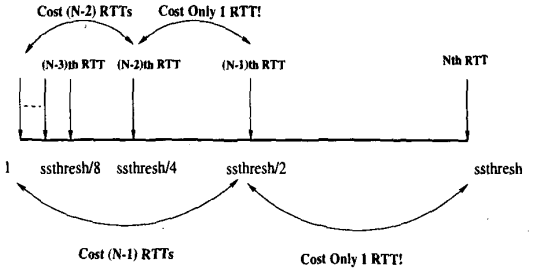


Figure 1. Illustration of the Aggressiveness of Slow-start Algorithm

When the congestion window size is equal to a half of *ssthresh*, it will reach the value of *ssthresh* at the next round-trip time. It could make the congestion window size much larger than the actual bandwidth-delay product at the next round-trip time if *ssthresh* is over-estimated, which often causes multiple packet losses from a window of data. This fact is illustrated in Figure 1. So, intuitively it is reasonable to slow down the rate of increasing the congestion window size at this point, allowing the sender to spend a little more time to reach *ssthresh* while approaching the actual equilibrium of the connection.

4 Smooth-Start

The proposed variant of Slow-start does not make more accurate estimation or dynamically adjust the estimation according to the changes. Rather, it is a more graceful rampup of congestion window size in the Slow-start phase, so that packet losses may be reduced significantly. The rationale behind Smooth-start is that the default value of *ssthresh* is often larger than the actual capacity of pipe size, and the estimated *ssthresh* is often over-estimated due to ACK compression.

4.1 Smooth-start: A New Variant of Slow-start

As Figure 1 shows, the problem with Slow-start is that *cwnd* takes i RTTs to reach $\frac{ssthresh}{2^{n-i}}$ from 1, but only 1 RTT to reach $\frac{ssthresh}{2^{n-(i+1)}}$ from $\frac{ssthresh}{2^{n-i}}$. This acceleration at the end could cause congestion and packet loss if *ssthresh* is set to default or the available bandwidth is over-estimated. Here, we assume that *cwnd* needs n RTTs to reach *ssthresh* from 1.

Figure 2 shows the dynamics of the congestion window size in Slow-start and Smooth-start if no congestion occurs.

In Slow-start, the congestion window size grows exponentially until it reaches *ssthresh* which is set to 32 Kbytes, and then grows linearly in the Congestion-Avoidance phase. In Smooth-start, the congestion window size also grows exponentially until it reaches the separator of Smooth-start which is set to *ssthresh*/4 in this case, then grows at a slower exponential rate in the Smooth-start period. After reaching *ssthresh*, it linearly enters the Congestion-Avoidance phase.

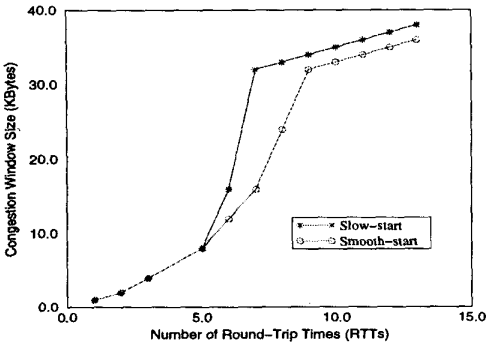


Figure 2. Illustration of Slow-start and Smooth-start algorithms

b The key idea of Smooth-start is to add necessary lag points between $\frac{ssthresh}{2^{n-i}}$ and $\frac{ssthresh}{2^{n-(i+1)}}$ to dampen the effect of exponentially increase. In Smooth-start, there are two sub-phases defined by a chosen separator *smsthresh* (in Figure 2, *ssthresh/4* is chosen as a separator). In the first sub-phase, Smooth-start behaves the same way as Slow-start. So, we call this period the *Slow-start* period. In the second sub-phase, called the *Smooth-start* period, *cwnd* is incremented only upon receipt of two or more ACKs. So, at every RTT *cwnd* increases by a factor of 1.5 or less. The number of RTTs needed for *cwnd* to reach from $\frac{ssthresh}{2^{n-i}}$ to $\frac{ssthresh}{2^{n-(i+1)}}$ in the *Smooth-start* period depends on the choice of the increasing gradient of the congestion window size. Two types of increasing gradient are introduced: *coarse-grained* and *fine-grained*.

4.1.1 Coarse-grained Probe

In the case of the coarse-grained probe, one *lag point* is inserted between $\frac{ssthresh}{2^{n-i}}$ and $\frac{ssthresh}{2^{n-(i+1)}}$, where a lag point represents an additional round-trip interval. In the first RTT after congestion window size reached $\frac{ssthresh}{2^{n-i}}$, two ACKs trigger one increment of *cwnd*. In the second RTT, three ACKs trigger one increment of *cwnd*. Then two RTTs are needed to increase *cwnd* from $\frac{ssthresh}{2^{n-i}}$ to $\frac{ssthresh}{2^{n-(i+1)}}$. The detailed calculation is shown below.
At the end of the first RTT in the Smooth-start sub-phase,

$$CWND = \frac{ssthresh}{2^{n-i}} + \frac{1}{2} \times \frac{ssthresh}{2^{n-i}} = \frac{3 \cdot ssthresh}{2^{n-i+1}}$$

At the end of the second RTT,

$$CWND = \frac{3 \cdot ssthresh}{2^{n-i+1}} + \frac{1}{3} \times \frac{3 \cdot ssthresh}{2^{n-i+1}} = \frac{ssthresh}{2^{n-(i+1)}}$$

Therefore, two RTTs are required to increase *cwnd* from $\frac{ssthresh}{2^{n-i}}$ to $\frac{ssthresh}{2^{n-(i+1)}}$.

4.1.2 Fine-grained Probe

In the case of the fine-grained probe, two lag points are inserted between $\frac{ssthresh}{2^{n-i}}$ and $\frac{ssthresh}{2^{n-(i+1)}}$. In the first RTT after *cwnd* reached $\frac{ssthresh}{2^{n-i}}$, three ACKs trigger one increment of *cwnd*. In the second RTT, four ACKs trigger one increment of the congestion window size, and in the third RTT, five ACKs trigger one increment of the congestion window size. The detailed calculation is as follows:
At the end of the first RTT,

$$CWND = \frac{ssthresh}{2^{n-i}} + \frac{1}{3} \times \frac{ssthresh}{2^{n-i}} = \frac{4 \cdot ssthresh}{3 \times 2^{n-i}}$$

At the end of the second RTT,

$$CWND = \frac{4 \cdot ssthresh}{3 \times 2^{n-i}} + \frac{1}{4} \times \frac{4 \cdot ssthresh}{3 \times 2^{n-i}} = \frac{5 \cdot ssthresh}{3 \times 2^{n-i}}$$

At the end of the third RTT,

$$CWND = \frac{5 \cdot ssthresh}{3 \times 2^{n-i}} + \frac{1}{5} \times \frac{5 \cdot ssthresh}{3 \times 2^{n-i}} = \frac{ssthresh}{2^{n-(i+1)}}$$

Therefore, a total of three RTTs are required to increase the congestion window size from $\frac{ssthresh}{2^{n-i}}$ to $\frac{ssthresh}{2^{n-(i+1)}}$.

In general, if we start with k ACKs triggering one increment of the congestion window size at the first RTT, and $(k+i)$ ACKs triggering one increment of *cwnd* at the next i-th RTT, we need k RTTs to increase *cwnd* from $\frac{ssthresh}{2^{n-i}}$ to $\frac{ssthresh}{2^{n-(i+1)}}$. We call k the *grain number*. The number of lag points inserted between $\frac{ssthresh}{2^{n-i}}$ and $\frac{ssthresh}{2^{n-(i+1)}}$ is $k-1$. In the case of coarse- or fine- grained probe that is studied in the rest of the paper, the grain number is set to 2 or 3.

4.2 Possible Separator

The general form of the separator of the *Slow-start* period from *Smooth-start* period is $ssthresh/2^d$, where d is called the *depth number*. The two candidates in our consideration are: $ssthresh/2$ and $ssthresh/4$, in which d is set to 1 and 2, respectively. We do not consider $d \geq 3$ because the separator gets too small. In the current implementation of TCP Slow-start, the default value of *ssthresh* is set to 64 Kbytes. If we choose $d = 3$, *smsthresh* becomes 8 Kbytes. Considering the proposed initial window size of 4 Kbytes, the Slow-start period will only last for one RTT. In many cases (except for heavy congestion), this would degrade the performance of a TCP connection as it would take too long to reach the optimal operating point.

The rule for inserting lag points is illustrated in Figure 3 in the case of the separator equal to $ssthresh/2$, which contains two graphs representing the rules for the coarse- and fine- grained probes, respectively.

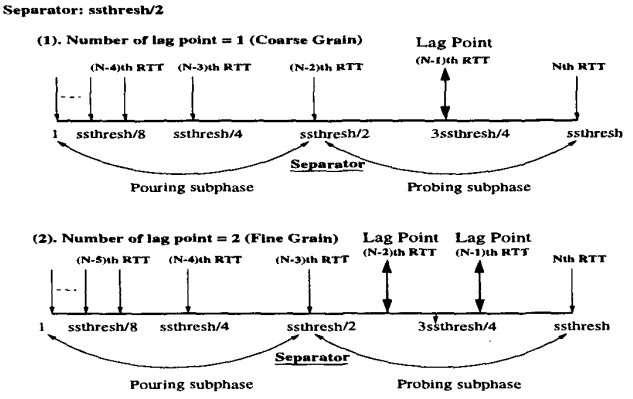


Figure 3. Separator is ssthresh/2

4.3 Overhead of the Smooth-start

The grain number k and the depth number d — both are non-negative integers — are two important parameters of the Smooth-start algorithm. The grain number k controls the probe gradient, and the depth number d controls the probe depth. The Smooth-start becomes more conservative as k or d increases. The overhead of Smooth-start depends on the underlying probe strategy, which is determined by k and d. In general, if the separator is set to $\frac{ssthresh}{2^d}$ and the grain number to k, then the total number of extra RTTs introduced by Smooth-start *ExtraRTTs* is:

$$ExtraRTTs = d \times (k-1)$$

Note that if k is set to 1 and d to 0, then the Smooth-start becomes the Slow-start. To some extent, The Slow-start can be viewed as a special case of Smooth-start. In this paper, the grain number k is limited to the set of $\{2,3\}$ and the depth number d to the set of $\{1,2\}$.

Since only two choices of grain and depth numbers, respectively, are considered in this paper, we studies a total of 4 probe strategies. The number in Table 1 indicates how many lag points are inserted in each case. The number of lag points corresponds to the number of extra RTTs introduced by the Smooth-start for *cwnd* to reach *ssthresh*.

Separator	Coarse-grained Probe	Fine-grained Probe
ssthresh/2	1	2
ssthresh/4	2	4

Table 1. Number of lag points inserted by the probing strategies

5 Evaluation of Smooth-start

5.1 The Simulation Setup

We evaluated the Smooth-start algorithm by implementing it in the LBNL network simulator *ns* [11]. The simulation network topology used is shown in Figure 4, where S_i represents a sender host and K_i a receiver host. $R1$ and $R2$ represent two finite-buffer gateways. The links are labeled with their bandwidth and one-way propagation delay. Different connections (from S_i to K_i) share a common bottleneck of 1.5 Mbps. The buffer size at each gateway R_i is set to 25 packets in each experiment. The type of data traffic in our simulation is FTP. The receiver sends an ACK for every data packet received. For the convenience of presentation, we assume that all window sizes are measured in number of fixed-size packets. For

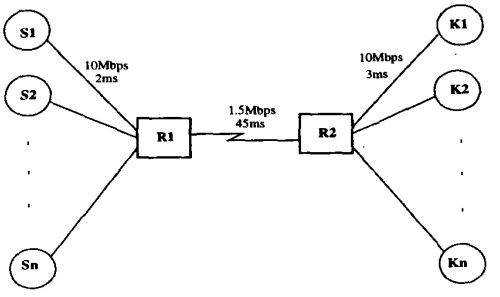


Figure 4. Simulation topology used for Smooth-start experiments

the constant-load experiment, drop-tail gateways with FIFO service are used. However, for the changing-load experiment, RED gateways [6], [18] are used instead. Although we have modified different versions of TCP like Tahoe, Reno, New-Reno and SACK to be equipped with the Smooth-start, only the simulation results of TCP Reno are presented here, because the key difference among various TCP versions lies in their own packet-loss recovery mechanisms. All TCP versions use the Slow-start at the start of a connection, and/or after a retransmission timeout, and hence they have similar behaviors during the Slow-start phase. Another reason is that TCP Reno is built on UNIX BSD4.3, which is the most widely-used version of TCP.

In later sections where we show the simulation results, *Coarse* stands for the coarse-grained probe, and *Fine* for the fine-grained probe. The number 2 or 4 indicates that the chosen separator is *ssthresh/2* or *ssthresh/4*, respectively.

5.2 Constant-Load Experiment with Drop-tail Gateways

In the constant-load experiment, the cross-traffic is constant and the TCP connection we measured has a fixed share of 1 Mbps and 8 buffer units. Thus, the bandwidth-delay product of each TCP connection is:

$$BWP = 1\,Mbps \times 2(45+5)\,ms = 100\,Kbits = 12.5\,Kbytes$$

Since each data packet size is 1024 bytes, the actual bandwidth-delay product is approximately 12 packets. The

total number of packets that can be in flight is 20 plus the number of packets in the buffer of the TCP connection. The file size transferred by this TCP connection is 60 Kbytes.

rwnd is varied to be 18, 20, 24, 36 and 64. Since TCP sets *ssthresh* to be the minimum of the default value and *rwnd*, varying *rwnd* effectively changes the ratio of *ssthresh* to the actual pipe size. We need to consider five different cases: (1) *ssthresh* is smaller than the actual pipe size; (2) *ssthresh* is equal to the actual pipe size; (3) *ssthresh* is larger than the actual pipe size; (4) *ssthresh* is much larger than the actual pipe size; (5) *ssthresh* equals the default setting, which is more than three times larger than the actual pipe size. The case where TCP window size is much smaller than the actual pipe so that no congestion occurs, will be discussed in Section 5.5.

The packet losses of a TCP connection using the Slow-start and Smooth-start are plotted in Figures 5 and 6; Figures 5 shows the reduction in packet losses when the separator = *ssthresh/2*, while Figure 6 shows the reduction in packet losses when the separator = *ssthresh/4*.

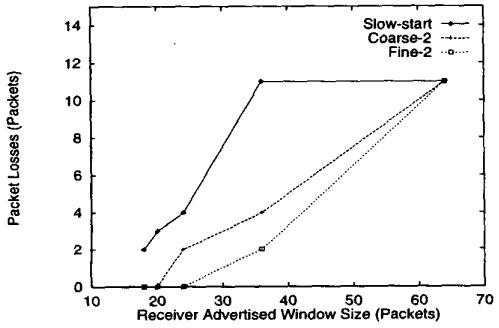


Figure 5. Separator = ssthresh/2

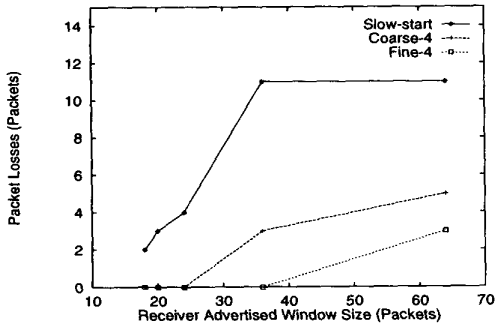


Figure 6. Separator = ssthresh/4

Obviously, the Smooth-start significantly reduces packet losses in all cases when the separator is *ssthresh/4* and in most cases when the separator is *ssthresh/2*. It is not surprising to see that if the separator of Smooth-start is chosen to be *ssthresh/2* and the *ssthresh* is set to the default value[1], the number of packet losses in the Smooth-start is the same as that in the Slow-start, because the separator *ssthresh/2* is larger than the actual pipe size. Thus, before *cwnd* reaches *ssthresh/2* to begin the Smooth-start period, multiple packets have already been dropped due to the buffer overflow during the Slow-start period. This is the main reason why we advance the probing phase by choosing the separator to be *ssthresh/4*. It also shows why an inaccurate initial *ssthresh* estimator is still more helpful than the default setting.

[1] It is three times larger than the actual pipe size.

To clearly illustrate the dynamics of a TCP connection, the standard technique of TCP sequence number plots is used. The graphs were generated by tracing packets entering and departing router R1. Packets are numbered starting with packet 0. Figure 7 presents the dynamics of the TCP connection with different *rwnd* and probe strategies. It shows us that Smooth-start has fewer packet drops, fewer timeouts, and consistently higher throughput than Slow-start.

Because TCP Reno performs poorly in recovering from multiple packet losses, the transfer latency is also greatly reduced. However, reduction of the transfer latency depends on the loss recovery mechanism of different TCP versions. For TCP New-Reno and SACK, the reduction of transfer latency is not as significant as that of TCP Reno.

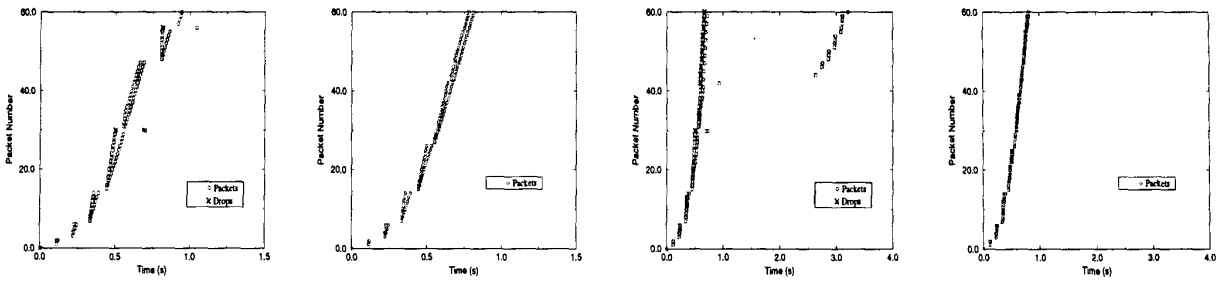
Note when the initial *ssthresh* is set close to the actual pipe size, the Smooth-start still can make significant performance improvements over the Slow-start, which indicates that an accurate and timely *ssthresh* estimation cannot entirely eliminate the performance degradation caused by the Slow-start. The TCP performance degradation due to the inappropriate approach that Slow-start takes in reaching *ssthresh* can only be remedied by the Smooth-start. An accurate estimation of initial *ssthresh* can significantly improve TCP performance, but is not a panacea to remedy all TCP start-up problems.

5.3 Changing-Load Experiment with RED Gateways

The setting of the changing-load experiment is similar to the constant-load case, except for the fact that RED gateways replace drop-tail gateways and the traffic load is periodically changed. Twelve TCP connections share the common bottleneck of 1.5 Mbps. The first connection starts at time 0.5. After that, once every 0.5 second, we start a new TCP connection. The probe strategy of Smooth-start is *Fine-4* and *rwnd* is 64 packets. In our experiment, the RED gateway notifies early the congestion to connections by dropping packets.

The simulation results are shown in two parts. The first part is about the queueing behaviors at the RED gateway, indicating the degree of burstiness of the incoming TCP traffic. The second part shows the traffic dynamics of one TCP connection, which starts at 2.0s and sends 1 Mbytes of data.

The dynamics of the actual queue size at the RED gateway is plotted in Figure 8. Clearly, the Smooth-start greatly reduces the fluctuation of actual queue size and the frequency of



(a) Slow-start, rwnd = 18 (b) Coarse-2, rwnd = 18 (c) Slow-start, rwnd = 36 (d) Fine-4, rwnd = 36

Figure 7. Simulation results for TCP Reno with different rwnd and probe strategies

the buffer overflow. By replacing the Slow-start with Smooth-start, the burstiness of the incoming traffic is reduced, and hence, the pressure upon the buffer of the gateway is reduced.

Figure 9 shows the traffic dynamics of the TCP connection, which starts at 2.0s. As expected, the Smooth-start has consistently higher throughput, less packet losses, and less transfer latencies than the Slow-start.

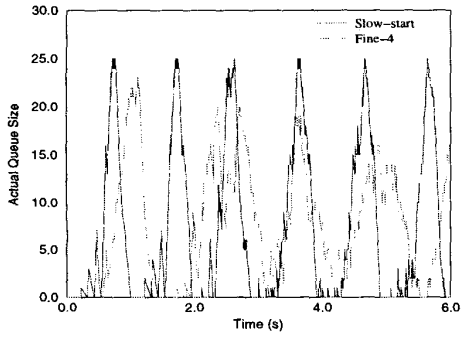


Figure 8. Dynamics of actual queue size at the RED gateway

5.4 Integration with *ssthresh* Estimator

As the Smooth-start does not require an *ssthresh* estimator [7], integration of Smooth-start and *ssthresh* estimator yields a better performance. The key point here is that the estimated *ssthresh* is no worse than, and is frequently better than, an arbitrary default. The simulation results are shown in table 2. In the same scenario where 6 packets are dropped, by employing Smooth-start, the number of packet losses is reduced to 2. By integrating the Smooth-start with an *ssthresh* estimator, only one packet is dropped.

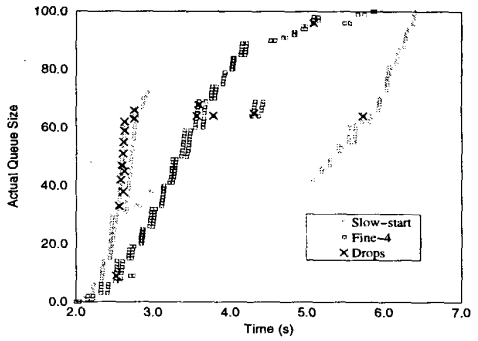


Figure 9. Illustration of the dynamics of a TCP connection

Slow-start	Smooth-start	Smooth-start /w Estimator
6	2	1

Table 2. Reduction of packet losses

5.5 Side-Effect of Smooth-start

If no congestion occurs, the performance of Smooth-start is worse than that of Slow-start. However, since the transmission rate still increases exponentially during the Smooth-start period, the degradation of TCP performance is not significant. The worst case happens if the TCP connection is closed exactly when the congestion window size *cwnd* reaches *ssthresh*. If the transmission of data lasts after reaching *ssthresh*, the degradation of TCP performance caused by the Smooth-start will become more trivial. Table 3 shows the percentage of the utilized bandwidth of a TCP connection if no congestion occurs.

Moreover, the proposed larger initial window size and TCP Fast-start, which happen at the first RTT of the Slow-start period, compensate the slower growth rate of the congestion window during the Smooth-start period. According to

the evaluation report in [3], the total savings provided by the proposed initial window size is up to 3 RTTs if the receiver acknowledges every received packet. By integrating the Smooth-start with the larger initial window, the amount of time required for *cwnd* to reach *ssthresh* can be computed as follows:

$$RTT \times \log_2 ssthresh + d \times (k-1)RTT - 3RTT.$$

Therefore, for the probe strategy *Fine-4*, the Smooth-start takes only one more RTT to reach *ssthresh*; but for the other probe strategies considered in this paper, the Smooth-start takes less RTTs to reach *ssthresh* if the larger initial window is employed.

Duration	Slow-start	Coarse-2	Fine-2
2s	49.8%	47.2%	44.6%
4s	73.6%	72.2%	70.8%
6s	82.7%	81.8%	80.9%
8s	86.8%	86.1%	85.4%

Table 3. The percentage of utilized bandwidth of a TCP connection

Note that in the environment of drop-tail gateways, the Smooth-start only postpones congestion, instead of eliminating it. Finer-grained probe strategies are found to be able to lead to more packet losses and reduced throughput for large file transfers, which is counter-intuitive. The reason for this is two-fold. Fisrt, the drop-tail gateway does not drop any packet until the buffer is full. Second, the finer-grained probe strategies delay the buffer overflow, and hence the congestion signal indicated by a packet loss.

Late arrivals of the congestion signal cause the TCP senders to inject more traffic load into the network. Once the bottleneck runs out of buffer space, a severer congestion occurs. However, with RED gateways, this phenomenon does not hold any more. The RED gateway signals the TCP senders of potential congestion before the buffer gets full. The simulation results shown in Section 5.3 confirms this, where Fine-4 is used and a 1 Mbyte file is transferred.

6 Conclusion

We proposed and evaluated a new variant of the Slow-start, called the Smooth-start, to improve the TCP start-up performance. The Smooth-start smoothes the transition between the Slow-start and Congestion-Avoidance phases. Two important parameters, grain number k and depth number d, are introduced for the Smooth-start, which determine the conservativeness of Smooth-start. Several possible probe strategies of the Smooth-start are evaluated through simulation. The simulation results demonstrated the inherent superiority of Smooth-start to Slow-start. With the Smooth-start algorithm, the chance of multiple packet losses from the same window of data is significantly reduced and the effective throughput is greatly improved. The Smooth-start can also be easily implemented and deployed since it requires only the sender side to be modified.

References

[1] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window", *INTERNET DRAFT*, draft-floyd-incr-init-win-03.txt, April 1998.

[2] M. Allman and V. Paxson, "On Estimating End-to-End Network Path Properties", *Proceedings of ACM SIGCOMM'99*, Cambridge, MA, pp. 263-274, August 1999.

[3] M. Allman, C. Hayes, and S. Ostermann, "An Evaluation of TCP with Larger Initial Windows", *ACM Computer Communication Review*, Vol. 28 No. 3, pp 41-52, July, 1998.

[4] B. Braden et al, "Recommendations on Queue Management and Congestion Avoidance in the Internet", *INTERNET DRAFT*, draft-irtf-e2e-queue-mgt-00.txt, March 1997.

[5] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Technique for Congestion Detection and Avoidance", *Proceedings of ACM SIGCOMM'94*, London, UK, pp. 24-35, August 1994.

[6] S. Floyd and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, pp. 397-413, August 1993.

[7] J. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *Proceeding of ACM SIGCOMM'96*, Stanford, CA, pp. 270-280, August 1996.

[8] V. Jacobson, "Congestion Avoidance and Control", *Proceedings of ACM SIGCOMM'88*, Stanford, CA, pp. 314-329, August 1988.

[9] S. Keshav, "A Control-Theoretic Approach to Flow Control", *Proceedings of ACM SIGCOMM'91*, Zurich, Switzerland, pp. 3-15, September 1991.

[10] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Option", Internet Draft, work in progress, May 1996.

[11] S. McCanne and S. Floyd, ns-LBNL Network Simulator. Obtain via: http://www-nrg.ee.lbl.gov/ns/.

[12] V. N. Padmanabhan and R. H. Katz, "TCP Fast Start: A Technique for Speeding Up Web Transfers", *Proceedings of IEEE GLOBECOM'98*, Sydney, Australia, November 1998.

[13] J. Postel, Transmission control protocol, Request for Comments 793, DDN Network Information Center, SRI International, September 1981.

[14] S. Seshan, M. Stemm, and R. H. Katz, "SPAND: Shared Passive Network Performance Discovery", *Proceedings of USITS'97*, Monterey, CA, December 1997.

[15] S. Shenker, L. Zhang, and D. D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm", *ACM Computer Communication Review*, Vol. 20, No. 4, pp. 30-39, October 1990.

[16] W. R. Stevens, *TCP/IP Illustrated*, volume 1. Addison-Wesley Publishing Company, 1994.

[17] K. Thompson, G. J. Miller, and R. Wilder, "Wide-Area Internet Traffic Patterns and Characteristics", *IEEE Network*, Vol. 11, No. 6, pp. 10-23, November/December 1997.

[18] H. Wang and K. G. Shin, "Refined Design of Random Early Detection Gateways", *Proceedings of IEEE GLOBECOM'99*, Rio de Janeiro, Brazil, December, 1999

[19] Y. Zhang, L. Qiu, and S. Keshav, "Optimizing TCP Start-up Performance", Cornell CS Technical Report, February 1999.

[20] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two Way Traffic", *Proceedings of ACM SIGCOMM'91*, Zurich, Switzerland, pp. 133-148, September 1991.